A Risk Taxonomy Proposal for Software Maintenance

Kénia P. Batista Webster, Káthia M. de Oliveira, Nicolas Anquetil
UCB - Catholic University of Brasilia
SGAN 916 Módulo B - Av. W5 Norte
Brasilia - DF - 70.790-160, Brazil
{kathia,anquetil}@ucb.br

Abstract

There can be no doubt that risk management is an important activity in the software engineering area. One proof of this is the large body of work existing in this area. However, when one takes a closer look at it, one perceives that almost all this work is concerned with risk management for software development projects. The literature on risk management for software maintenance is much scarcer.

On the other hand, software maintenance projects do present specificities that imply they offer different risks than development. This suggests that maintenance projects could greatly benefit from better risk management tools. One step in this direction would be to help identifying potential risk factors at the beginning of a maintenance project. For this, we propose a taxonomy of possible risks for software management projects. The ontology was created from: i) an extensive survey of risk management literature, to list known risk factors for software development; and, ii) an extensive survey of maintenance literature, to list known problems that may occur during maintenance.

1 Introduction

Risk management for software is considered as one of foremost best practices to be applied in software projects [1, 5]. As such it received lot of attention and there are numerous publications on risk management (e.g. [3, 19, 17, 21, 26, 28]), risk identification methods (e.g. [6, 20, 25]), etc. However, despite this profusion of work, there has been very little study on risk management for software maintenance projects [8].

Although it is the activity most practiced in industry (up to 90% of the work according to some research [29, 30]), software maintenance receives less attention than other activities in software engineering1. This is also true for risk management [8]. Maintenance suffers from an erroneous view that equates it to bug correction. However, studies (for example cited in[15, 30]) show that corrective maintenance accounts for only (approximately) 20% of all maintenance, whereas the larger part (up to 50%) is due to evolutive maintenance (implementation of new requirements). Actually, maintenance is an unavoidable activity required to keep systems synchronized with the reality they are modeling, a reality that changes continuously.

Maintenance also presents specificities that set it apart from software development [8, 15]. In risk management, this implies that existing methodologies for identification, evaluation and management of risk factors, which were created for development projects, may not be adequate for maintenance projects. On the other hand, reports suggest that risk management would contribute to improve the quality and efficiency of software evolution [7, 34].

Considering the importance of both software maintenance and risk management for the industry we started a project on Risk Management for Software Maintenance. A first goal in this project was to establish the means to identify potential risks of a new maintenance project. A well accepted method for this is to use some kind of check list or risk taxonomy. In this article, we propose such a taxonomy for maintenance project risk factors. This taxonomy was established from two literature researches, both on software development risk factors and software maintenance known problems.

The organization of the article is the following: First, in Section 2 we review some basic concepts of risk management for software projects. Section 3 discusses software maintenance, its importance and typical prob-

1 For example, Grubb and Takang state that “maintenance models are neither so well developed nor so well understood as models for software development.” [15, p.71]
lems it may face. Then, in Section 4, we present the methodology we followed to create the taxonomy of risk factors for software maintenance. This taxonomy is presented and discussed in Section 5. Finally, we present our conclusion and future work in Section 6.

2 Risks in Software Projects

A risk, in a software project, is an event which occurrence is uncertain and that could impact negatively the project if it should occur [18, 19, 26, 34]. The definition highlights two intrinsic properties of risks:

Uncertainty: A risk is defined as an uncertain event, when it does happen, it is no longer a risk but becomes a problem.

Loss: A risk is a (possible) event which would damage in some way the project if it should happen.

In [20], Keil et al. studied various systems after delivery and established that many problems encountered could have been avoided if risk management had been applied during the projects. To increase the chances of success of a software project, one must constantly monitor possible risk factors, foresee possible solution should they occur, prepare for the possible occurrence of the risk, detect such occurrence when it happens, evaluate the severity and actual impact of the event, and apply the needed correction.

There is no unique definition on how one should go about managing risks, however, in the various approaches proposed (e.g. [3, 9, 16, 19, 14, 28, 35, 10]) two activities arise as fundamental:

Identification: Risk identification consists in enumerating all possible risks for a project (before they occur) [19]. There are many ways to identify risk factors (e.g. brainstorming session, fishbone diagram, etc.) but the most cited appears to be the use of a taxonomy of possible risks from which one identifies those that may apply to a particular project.

Analysis: Risk analysis consists in quantifying each identified risk in terms of its probability of materializing and its potential impact [11, p.20]. Using a matrix of probability and impact, risks may be classified from critical (high probability and high impact), to marginal (low probability and low impact). Table 1 illustrates an example of such a matrix.

In this paper, we propose a taxonomy of risks for software maintenance as a mean to facilitate the identification of risk factors in maintenance project.

Table 1. Example of a probability/impact matrix to classify risks

<table>
<thead>
<tr>
<th>Impact</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>critical</td>
</tr>
<tr>
<td>medium</td>
<td>critical</td>
</tr>
<tr>
<td>low</td>
<td>medium</td>
</tr>
<tr>
<td></td>
<td>marginal</td>
</tr>
</tbody>
</table>

In the next section we will summarize some important facts on software maintenance to illustrate the importance of this activity. We will also comment on the necessity and current scarcity of research on risk management for maintenance.

3 Software Maintenance

Maintenance is traditionally defined as any modification made on a system after its delivery. Studies show that software maintenance is, by far, the predominant activity in software engineering (90% of the total cost of a typical software [30, 36]). It is needed to keep software systems actual and useful. Any software system reflects the world within which it operates. When this world changes, the software needs to change accordingly. Lehman’s first law of software evolution (law of continuing change, [22]) is that “a program that is used undergoes continual change or becomes progressively less useful”. Maintenance is mandatory, one simply cannot ignore new laws or new functionalities introduced by a concurrent. Programs must also be adapted to new computers (with better performances) or new operational systems.

Maintenance differs from development for a number of reasons, including:

Lack of documentation: Developers may rely on the fact that somebody (hopefully close by) knows one specific portion of the system, in the best cases there may even be a clear documentation. Maintainers must often work from the source code to the exclusion of any other source of information.

Event driven: Development is requirement driven, one specifies the requirements and then plan their orderly implementation. Maintenance is event driven [30], external events require the modification of the software. There is much less opportunity for planning.

Obsolete techniques: Developers may organize the system as best suits the requirements. In the best
cases, they may even be able to choose the programming paradigm and language, the hardware platform, etc. Maintainers must cope with choices made by others in the past. The programming language may be old (e.g. COBOL), the system architecture may not fully support the modification they need to implement, and it may even be so obfuscated by past modifications that there is no longer any clear architecture.

**Differing processes:** The process of maintenance is different in that it requires new activities up-front which do not exist in development. For example, maintenance requires to analyze in depth the system to be modified before any analysis of the task at hand starts\(^2\). Because of this, the bulk of the effort in a maintenance project is applied at the beginning of the project, whereas it is applied towards the end of a development project (during the implementation and test\(^3\)) [15]. In development, the initial activity (requirement elicitation) may be difficult, but it typically requires less effort than the implementation because much less details are involved.

**System in operation:** During maintenance, the system is in operation, this may significantly increase the difficulty of altering the system while maintaining it operational.

These characteristics make maintenance an activity quite different from software development. They also imply that risk factors for software maintenance need not be the same as for software development [8]. This is why we were led to define a taxonomy of risk factors for software maintenance. The next section will explain the methodology we followed to create the taxonomy. The taxonomy itself is presented in Section 5.

## 4 Creating the Taxonomy

If risk management for software maintenance may be performed the same way as for software development, the risks encountered are not always the same (see previous section), and even when they are the same, they may have a different importance (for example a stronger impact) because of the differing conditions in which maintenance is performed. As was explained in

---

\(^2\)The analysis of the system is made even more difficult by the usual lack of documentation on the system.

\(^3\)In many maintenance projects (for example of legacy software), the test activity is crippled by the lack of previous test cases, the lack of detailed understanding of how the system should behave, and the pressure for a rapid delivery of the change.

---

In this paper, we propose a taxonomy of risk factors for software maintenance projects. This work was necessary as we found little work listing risk factors for software maintenance. For example, although Charette *et al.* [8] do state that risk factors for software maintenance are different, they use the taxonomy of risk factors proposed by Carr *et al.* [6] which focuses software development.

### Table 2. References used to build the taxonomy

<table>
<thead>
<tr>
<th>Reference</th>
<th># of risks proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SOFTWARE DEVELOPMENT RISKS</strong></td>
<td></td>
</tr>
<tr>
<td>[37] Soeiro 1999</td>
<td>18</td>
</tr>
<tr>
<td>[17] Houston <em>et al.</em> 2001</td>
<td>30</td>
</tr>
<tr>
<td>[27] Mizuno <em>et al.</em> 2000</td>
<td>23</td>
</tr>
<tr>
<td>[38] Sommerville 2001 (chap. 4)</td>
<td>14</td>
</tr>
<tr>
<td>[35] Schawibe 2002</td>
<td>21</td>
</tr>
<tr>
<td>[18, 19] Jalote 1999 &amp; 2002</td>
<td>10</td>
</tr>
<tr>
<td>[32] Pressman 2001</td>
<td>10</td>
</tr>
<tr>
<td>[23] Leopoldino 2004</td>
<td>32</td>
</tr>
<tr>
<td>[12] Farias 2002</td>
<td>15</td>
</tr>
<tr>
<td>[41] Williams <em>et al.</em> 1997</td>
<td>10</td>
</tr>
<tr>
<td><strong>SOFTWARE MAINTENANCE RISKS</strong></td>
<td></td>
</tr>
<tr>
<td>[29] Pfleeger 2001</td>
<td>12</td>
</tr>
<tr>
<td>[8] Charette 1997</td>
<td>7</td>
</tr>
<tr>
<td>[34] Schneidewind 2002</td>
<td>7</td>
</tr>
<tr>
<td>[31] Pigoski 2001</td>
<td>7</td>
</tr>
<tr>
<td>[33] Schneberger 1995</td>
<td>1</td>
</tr>
<tr>
<td>[38] Sommerville 2001 (chap. 26, 27)</td>
<td>9</td>
</tr>
</tbody>
</table>

Our taxonomy is based on adaptation of risk fac-
risk factors (for maintenance or development) of 473 risks (382+91), therefore we decided to consolidate this list into a shorter one (the various steps of the consolidation are illustrated in Figure 1):

- First, we analyzed the list of 382 risk factors for software development to identify and remove possible duplicates. Table 3 presents an example of this integration. This resulted in a list of 198 risk factors.

- Second, we ignored all software development risk factors that were not referenced by at least two authors. The idea was to identify the most pertinent risks, eliminating those that were proposed by only one author, possibly in a very specific context. When an author reuses another one’s proposition unmodified (Boehm and Carr’s taxonomies were the most used by others), we did not count this as a second reference as we judged that the second author did not identify risk factors, but merely reused an existing work. Our list of risks was now down to 56 risk factors for software development.

- Third, we analyzed the list of 91 risk factors and problems for software maintenance to identify and remove possible duplicates. As there were much fewer risk factors for software maintenance, we did not filter them on the number of references, all risks were considered even if referenced by only one author. This gave us 54 risk factors.

- Fourth, we integrated the two lists of risk factors resulting from step 2 and 3 to establish a final list of 91 risk factors for maintenance.

To complete the creation of the taxonomy, we had to organize these 91 risk factors in categories. We chose to follow the organization scheme proposed by the SEI.

---

Table 3. Example of integration of similar risks

<table>
<thead>
<tr>
<th>Example of integration of similar risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performances do not meet expectations (including errors and low quality)⁴</td>
</tr>
<tr>
<td>Under-performance</td>
</tr>
<tr>
<td>Not meeting performance requirements</td>
</tr>
<tr>
<td>Link failure or slow performances</td>
</tr>
</tbody>
</table>

⇒ Not meeting performance requirements (ex.: under-performance)
taxonomy [6] since this is the work which was the most cited.

This taxonomy is organized in three levels: classes decomposed in elements, where the risk factors are classified. We reused Carr et al.’s three classes, adapting the names when necessary: Product Engineering (technical risks), Maintenance Environment (methodological risks), and Program Restrictions (organizational risks). As for the elements, we reused almost the same as proposed with a few modifications:

- Two elements (“Code and unit test” and “Integration and test”) of the Product Engineering class were merged into a more general element: “Test”. Several other taxonomies use this more general classification.
- As a consequence of this merge, the “Code and unit test” element was renamed “Code”.
- We added a “Legacy” element to the Product Engineering class. The SEI taxonomy is very much focused on development, and we felt the need to have an element reflecting some of maintenance’s specificities. In this case, the “Legacy” element is a recognition of the fact that a software system may come with defects that are independent of the current state of the maintenance organization but may impact a maintenance project.
- Element “Contract” was excluded from the Program Restriction class. After steps 1 and 2 (See Figure 1) this element had only one risk left. This risk was transferred to the “Program interface” element.
- Two elements (“Development process” and “Development system”) in the Maintenance Environment class were renamed to “Maintenance process” and “Maintenance system”, for obvious reasons.

5 Taxonomy of Risk Factors for Software Maintenance

As described in the preceding section our taxonomy follows the organization proposed by the SEI [6] with very similar classes and elements.

In this section, we present our taxonomy, listing the risk factors contained in each element of each class and the references where those risk factors were found.

The Product Engineering class consists of the activities required to maintain the product. Its risk factors are listed in Table 4. This class is divided in four elements:

- Requirements: Addresses the definition of what the system needs to do. It also addresses the feasibility of maintaining the system. This element contains six risk factors.
- Design: Addresses the translation of requirements into an effective design. This element contains eight risk factors.
- Code: Addresses the transformation of software designs into code that satisfies the requirements. This element contains nine risk factors.
- Test: Addresses the integrations of code units and the validation of the software product. This element contains two risk factors.

- Engineering specialties: Addresses the maintenance activities that may need specialized expertise. This element contains four risk factors.
- Legacy: Addresses some factors depending on the state of the system maintained (mainly non-code artifacts) prior to the maintenance. This element contains three risk factors.

The Maintenance Environment class is concerned with the project environment, where the software is being maintained. Its risk factors are listed in Table 5. This class is divided in five elements:

- Maintenance process: Lists the risk factors related to the definition, planning, documentation, suitability, enforcement, and communication of the methods and procedures used to maintain the product. This element contains seven risk factors.
- Maintenance system: Lists the risk factors related to the tools and supporting equipment used in product maintenance, such as CASE tools, simulators, compilers, and host computer systems. This element contains six risk factors.
- Management process: Lists the risk factors related to the planning, monitoring, and controlling of budgets and schedules; controlling factors involved in defining, implementing, and testing the product; the project managers experience in software maintenance, management, and the product domain; and the managers expertise in dealing with external organizations including customers, senior management, and other contractors. This element contains nine risk factors.
- Management methods: Lists the risk factors related to the methods, tools, and supporting equipment that will be used to manage and control the
### Table 4. Risk factors for the Product Engineering class

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Resource</th>
<th>Engineering specialties</th>
<th>Legacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Continuing stream of requirements changes [1, 4, 6, 8, 11, 12, 13, 17, 19, 25, 26, 32, 37, 38]</td>
<td>• Change may impact directly on other systems' functionalities [8]</td>
<td>• Lack of interactive agreement regarding requirement specifications between the customer and the developer [6, 27]</td>
<td>• System documentation nonexistent, incomplete or outdated [8, 24, 29, 30, 38]</td>
</tr>
<tr>
<td>• Requirements incompletely specified [6, 20]</td>
<td>• Change may impact directly on current systems' functionalities [8]</td>
<td>• Specifications change inadequate [6, 11, 12, 27, 29]</td>
<td>• Poor data integrity [24]</td>
</tr>
<tr>
<td>• Requirements unclear (ambiguous or imprecise) [6, 19, 25, 27]</td>
<td>• Large number of lines of code affected by change request [34]</td>
<td>• Unclear or misunderstood scope and/or objectives [1, 13, 23, 35]</td>
<td>• Data stored in differing or incompatible formats [38]</td>
</tr>
<tr>
<td>• Requirements misunderstood (do not reflect customer intentions) [1, 6, 13, 20, 25, 27]</td>
<td>• Change affect an area of the code that is critical to the project [34]</td>
<td>• Real-time and highly synchronized systems [29]</td>
<td></td>
</tr>
<tr>
<td>• Gold Plating (i.e. adding more features than is necessary) [1, 4, 13, 26]</td>
<td>• Large number of principal software functions affected by the change [34]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Performance requirements not met [11, 19, 37]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Design</th>
<th>Test</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Limited understanding [29, 31, 33]</td>
<td>• Inadequate test planning and preparation [6, 26, 29, 30, 34]</td>
<td>• Significant cost of repeating full testing on a piece of software [31]</td>
</tr>
<tr>
<td>• Unreliable hardware characteristics or unreliable vendor support [24, 29]</td>
<td>• Specifications change inadequate [6, 11, 12, 27, 29]</td>
<td>• Lack of interactive agreement regarding requirement specifications between the customer and the developer [6, 27]</td>
</tr>
<tr>
<td>• High level of complexity of the required change [6, 17, 25, 34]</td>
<td>• Real-time performances shortfalls [4, 6, 26]</td>
<td>• Specifications change inadequate [6, 11, 12, 27, 29]</td>
</tr>
<tr>
<td>• Developing the wrong user interface [4, 26, 37]</td>
<td>• Hardware constraints [4, 6, 24, 26, 34]</td>
<td>• Real-time and highly synchronized systems [29]</td>
</tr>
<tr>
<td>• Real-time performances shortfalls [4, 6, 26]</td>
<td>• New technology introduction [1, 13, 19, 20, 23, 25, 37]</td>
<td>• Real-time and highly synchronized systems [29]</td>
</tr>
<tr>
<td>• Hardware constraints [4, 6, 24, 26, 34]</td>
<td>• Low design quality (system’s components should be independent and cohesive) [15, 24, 29]</td>
<td>• Real-time and highly synchronized systems [29]</td>
</tr>
<tr>
<td>• New technology introduction [1, 13, 19, 20, 23, 25, 37]</td>
<td>• Low design quality (system’s components should be independent and cohesive) [15, 24, 29]</td>
<td>• Real-time and highly synchronized systems [29]</td>
</tr>
<tr>
<td>• Low design quality (system’s components should be independent and cohesive) [15, 24, 29]</td>
<td>• Change affect an area of the code that is critical to the project [34]</td>
<td>• Real-time and highly synchronized systems [29]</td>
</tr>
<tr>
<td>• Change affect an area of the code that is critical to the project [34]</td>
<td>• Large number of principal software functions affected by the change [34]</td>
<td>• Real-time and highly synchronized systems [29]</td>
</tr>
<tr>
<td>• Large number of principal software functions affected by the change [34]</td>
<td>• Change affect an area of the code that is critical to the project [34]</td>
<td>• Real-time and highly synchronized systems [29]</td>
</tr>
<tr>
<td>• Real-time and highly synchronized systems [29]</td>
<td>• Change may impact directly on other systems' functionalities [8]</td>
<td>• Lack of interactive agreement regarding requirement specifications between the customer and the developer [6, 27]</td>
</tr>
</tbody>
</table>

Product maintenance, such as monitoring tools, personnel management, quality assurance, and configuration management. This element contains 14 risk factors.

**Working environment:** Lists the risk factors related to the general environment within which the work will be performed, including the attitude of people and the level of cooperation, communication, and morale. This element contains six risk factors.

The Program Constraints class consists of the risks external to the project, factors that are outside the direct control of the project but can still have major effects on its success. Its risk factors are listed in Table 6. This class is divided in two elements:

**Resources:** Lists the risk factors related to the external constraints imposed on schedule, staff, budget, or facilities. This element contains five risk factors.

**Program Interfaces:** Lists the risk factors related to the external interfaces to customers, other contractors, corporate management, and vendors. This element contains 12 risk factors.

### 6 Conclusion

Risk management is considered a fundamental area of software engineering and project management. However, risk management for software maintenance projects has been little explored and this important activity suffers from a lack of tools to adequately identify and evaluate risk factors.

In this paper, we propose a taxonomy of risk factors for software maintenance. Risk factors were integrated from an extensive survey of literature on risk management for software development and on maintenance problems. We identified more than 400 risks in the literature from which we extracted 91 risk factors relevant to software maintenance projects. These 91 risk factors were classified according to the categorization of the risk taxonomy proposed by the SEI [6].

The current taxonomy is a first proposal, several
new references have been pointed to us by the reviewers of ICSM’05 and we will endeavor to update the taxonomy in the light of these new references. A first evaluation of the task showed that it may not bring many changes. We analyzed 47 risk factors listed in [39, p.50] and found them to be generally at a much lower abstraction level than our risks. This results in many of them being more specific than risks factors we already consider (e.g. “Efficiency” is a specific case of “Performance requirements not met”), or over specific (e.g. “No error message”).

The next step planned will be to propose an evaluation of the probability and impact of these risks for typical maintenance projects. This proposition will be based on the opinion and experience of maintenance managers. Based on this evaluation, the risks we propose could be prioritized. This would be useful to help maintenance managers getting gradually acquainted with the taxonomy by initially focusing on the risks that are typically most critical. It could also help to validate the effectiveness of the taxonomy by identifying risks that have typically little impact on software maintenance projects.

7 Acknowledgment

This work is part of a project, which is supported by the CNPq, an institution of the Brazilian government for scientific and technological development.

References


## Table 5. Risk factors for the Maintenance Environment class

### Maintenance process
- Inadequately selected maintenance model, process, methods or tools support [6, 15, 25, 30]
- Maintainability features not incorporated into the software effort [6, 31]
- Lacking or inadequate control of the maintenance process [6, 23, 27]
- Inadequate product control (ex.: product does not attend customer’s expectations) [6, 12]
- Excessive paperwork [17, 37]
- Lack of adherence to programming standards [24]
- Difficulties in adapting to a rapidly changing business environment [30]

### Maintenance system
- Insufficient capacity of the maintenance system (ex.: too few work-stations, insufficient processing power, ...) [6, 38]
- Lack of reliability in maintenance system (ex.: due to unavailable components or wrong functions and properties) [4, 6, 17, 26]
- Poor system support (ex.: training in use of the system, or resolution of problems by vendors) [6, 25]
- Failure when running the system [24]
- System hardware and software changes [24]
- Lack of support for re-engineering [30]

### Management process
- Inadequate planning [6, 25, 27, 35, 37]
- Ineffective definition of roles and responsibilities [6, 27, 35]
- Inexperience or inefficiency of management [6, 23, 25]
- Lack of interactions (communication) between management and stakeholders [6, 25]
- Failure to manage end user expectations [1, 20, 23, 24]
- Absence of leadership [25, 35]
- Lack of effective project management methodology [1, 13, 23, 25]
- Management priorities difficulties (ex.: focus on quick fixes in detriment of more robust solution) [29, 30]
- Organizational strategy based on erroneous analysis (ex.: Maintenance budget not based on needs analysis) [15, 30]

### Management methods
- Lack of monitoring of activities progresses [6, 25, 27]
- Inadequate maintainer training [6, 25, 30, 32, 37, 38]
- Inadequate quality assurance program [6, 35]
- Inadequate configuration control [6, 17, 25, 37]
- Continuing stream of maintenance scope or objectives changes [20, 23, 25]
- Inadequate or inaccurate time estimation [23, 35, 38]
- Inadequate or inaccurate cost estimation [17, 23, 30, 35]
- Inaccurate metrics [17, 37]
- Inadequate or inexact estimation [37, 27, 32, 38]
- Performance measurement difficulties [30]
- Contribution measurement difficulties [30]
- Large backlog [24, 30]
- Poor software and organization impact analysis (critical skills, documentation and process) [31]
- No programming standards [38]

### Working environment
- Lack of team spirit (ex.: internal conflicts requiring management intervention) [6, 12, 19, 25]
- Poor internal communication (ex.: lack of knowledge of the system mission or of its importance for program goals) [6, 25, 26]
- Lack of staff commitment, low morale (e.g. due to lack of recognition and respect) [6, 12, 17, 24, 23, 25, 27, 29, 30, 31]
- Lack of organizational maturity [17, 25]
- Low productivity of staff [12, 17, 24, 25, 29, 35]
- Lack of incentive to promote maintainability [38]

---


[33] S. Schneberger. Software maintenance in distributed computer environments: system complexity versus...
Table 6. Risk factors for the Program Restrictions class

<table>
<thead>
<tr>
<th>Resources</th>
<th>Program Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Unrealistic schedules (ex.: due to internal or external events) [1, 4, 6, 8, 11, 12, 13, 19, 24, 25, 26, 32, 37]</td>
<td>• Problems with contracts [6, 35, 37]</td>
</tr>
<tr>
<td>• Excessive schedule pressure [17, 25, 24]</td>
<td>• Problems with subcontractors [1, 6, 13, 37]</td>
</tr>
<tr>
<td>• Staff problems (ex.: lack of expertise, knowledge, skills, etc.) [1, 4, 6, 8, 13, 15, 17, 19, 20, 23, 24, 25, 26, 27, 30, 31, 32, 34, 38]</td>
<td>• Political factors (company, customer, associate contractors or subcontractor) [6, 12, 19, 25]</td>
</tr>
<tr>
<td>• Unrealistic budget (ex.: due to internal or external events) [1, 4, 6, 12, 13, 15, 25, 26, 31, 37]</td>
<td>• Lack of adequate user involvement and/or commitment [1, 13, 20, 23, 24, 37]</td>
</tr>
<tr>
<td>• Instability or lack of continuity in maintenance staffing [12, 15, 17, 23, 24, 25, 29, 30, 38]</td>
<td>• Lack of senior management commitment [1, 6, 13, 17, 20, 23, 24, 30]</td>
</tr>
<tr>
<td>• Any problem with customers (ex. slack approval, poor communication, lack of commitment, etc.) [6, 12, 17, 23, 25, 38]</td>
<td>• Conflict between user departments [13, 20, 23]</td>
</tr>
<tr>
<td>• Problems with contracts [17, 19, 31]</td>
<td>• User misunderstanding (ex.: they may provide maintainers with incomplete or misleading data [24, 29]</td>
</tr>
<tr>
<td>• Problems with subcontractors [1, 6, 13, 37]</td>
<td>• User resisting to changes [8, 15, 32]</td>
</tr>
<tr>
<td>• Political factors (company, customer, associate contractors or subcontractor) [6, 12, 19, 25]</td>
<td>• High turnover in user organization [24]</td>
</tr>
<tr>
<td>• Lack of adequate user involvement and/or commitment [1, 13, 20, 23, 24, 37]</td>
<td>• Failure to gain user commitment [13, 20]</td>
</tr>
<tr>
<td>• Lack of senior management commitment [1, 6, 13, 17, 20, 23, 24, 30]</td>
<td>• Inadequate user training [24]</td>
</tr>
</tbody>
</table>


